Interview Summary	Application No.	Applicant(s)	
	09/591,225	BALDWIN, DAVID ROBERT	
	Examiner	Art Unit	
	Aaron M. Richer	2628	

All participants (applicant, applicant's representative, PTO personnel):

(1) *Aaron M. Richer*. (3)_____.

(2) *Patrick Holmes*. (4)_____.

Date of Interview: *04 January 2007*.

Type: a)☒ Telephonic b)☐ Video Conference
c)☐ Personal [copy given to: 1)☐ applicant 2)☐ applicant's representative]

Exhibit shown or demonstration conducted: d)☐ Yes e)☐ No.
If Yes, brief description: _____.

Claim(s) discussed: *1-4,7,14,15 and 17*.

Identification of prior art discussed: *Kaiser (EP 0 766 177)*.

Agreement with respect to the claims f)☐ was reached. g)☒ was not reached. h)☐ N/A.

Substance of Interview including description of the general nature of what was agreed to if an agreement was reached, or any other comments: *Applicant's representative explained arguments presented in response to final rejection filed December 19, 2006. In particular, applicant's representative explained the differences between a TLB miss from the Kaiser reference and the page fault handled by the application. Examiner took arguments into consideration, but determined that more research into the broadest reasonable definition of a "page fault" was necessary* .

(A fuller description, if necessary, and a copy of the amendments which the examiner agreed would render the claims allowable, if available, must be attached. Also, where no copy of the amendments that would render the claims allowable is available, a summary thereof must be attached.)

THE FORMAL WRITTEN REPLY TO THE LAST OFFICE ACTION MUST INCLUDE THE SUBSTANCE OF THE INTERVIEW. (See MPEP Section 713.04). If a reply to the last Office action has already been filed, APPLICANT IS GIVEN A NON-EXTENDABLE PERIOD OF THE LONGER OF ONE MONTH OR THIRTY DAYS FROM THIS INTERVIEW DATE, OR THE MAILING DATE OF THIS INTERVIEW SUMMARY FORM, WHICHEVER IS LATER, TO FILE A STATEMENT OF THE SUBSTANCE OF THE INTERVIEW. See Summary of Record of Interview requirements on reverse side or on attached sheet.

Examiner Note: You must sign this form unless it is an Attachment to a signed Office action.

a m. r
Examiner's signature, if required

Summary of Record of Interview Requirements

Manual of Patent Examining Procedure (MPEP), Section 713.04, Substance of Interview Must be Made of Record
A complete written statement as to the substance of any face-to-face, video conference, or telephone interview with regard to an application must be made of record in the application whether or not an agreement with the examiner was reached at the interview.

Title 37 Code of Federal Regulations (CFR) § 1.133 Interviews
Paragraph (b)
In every instance where reconsideration is requested in view of an interview with an examiner, a complete written statement of the reasons presented at the interview as warranting favorable action must be filed by the applicant. An interview does not remove the necessity for reply to Office action as specified in §§ 1.111, 1.135. (35 U.S.C. 132)

37 CFR §1.2 Business to be transacted in writing.
All business with the Patent or Trademark Office should be transacted in writing. The personal attendance of applicants or their attorneys or agents at the Patent and Trademark Office is unnecessary. The action of the Patent and Trademark Office will be based exclusively on the written record in the Office. No attention will be paid to any alleged oral promise, stipulation, or understanding in relation to which there is disagreement or doubt.

The action of the Patent and Trademark Office cannot be based exclusively on the written record in the Office if that record is itself incomplete through the failure to record the substance of interviews.

It is the responsibility of the applicant or the attorney or agent to make the substance of an interview of record in the application file, unless the examiner indicates he or she will do so. It is the examiner's responsibility to see that such a record is made and to correct material inaccuracies which bear directly on the question of patentability.

Examiners must complete an Interview Summary Form for each interview held where a matter of substance has been discussed during the interview by checking the appropriate boxes and filling in the blanks. Discussions regarding only procedural matters, directed solely to restriction requirements for which interview recordation is otherwise provided for in Section 812.01 of the Manual of Patent Examining Procedure, or pointing out typographical errors or unreadable script in Office actions or the like, are excluded from the interview recordation procedures below. Where the substance of an interview is completely recorded in an Examiners Amendment, no separate Interview Summary Record is required.

The Interview Summary Form shall be given an appropriate Paper No., placed in the right hand portion of the file, and listed on the "Contents" section of the file wrapper. In a personal interview, a duplicate of the Form is given to the applicant (or attorney or agent) at the conclusion of the interview. In the case of a telephone or video-conference interview, the copy is mailed to the applicant's correspondence address either with or prior to the next official communication. If additional correspondence from the examiner is not likely before an allowance or if other circumstances dictate, the Form should be mailed promptly after the interview rather than with the next official communication.

The Form provides for recordation of the following information:
- Application Number (Series Code and Serial Number)
- Name of applicant
- Name of examiner
- Date of interview
- Type of interview (telephonic, video-conference, or personal)
- Name of participant(s) (applicant, attorney or agent, examiner, other PTO personnel, etc.)
- An indication whether or not an exhibit was shown or a demonstration conducted
- An identification of the specific prior art discussed
- An indication whether an agreement was reached and if so, a description of the general nature of the agreement (may be by attachment of a copy of amendments or claims agreed as being allowable). Note: Agreement as to allowability is tentative and does not restrict further action by the examiner to the contrary.
- The signature of the examiner who conducted the interview (if Form is not an attachment to a signed Office action)

It is desirable that the examiner orally remind the applicant of his or her obligation to record the substance of the interview of each case. It should be noted, however, that the Interview Summary Form will not normally be considered a complete and proper recordation of the interview unless it includes, or is supplemented by the applicant or the examiner to include, all of the applicable items required below concerning the substance of the interview.

A complete and proper recordation of the substance of any interview should include at least the following applicable items:
1) A brief description of the nature of any exhibit shown or any demonstration conducted,
2) an identification of the claims discussed,
3) an identification of the specific prior art discussed,
4) an identification of the principal proposed amendments of a substantive nature discussed, unless these are already described on the Interview Summary Form completed by the Examiner,
5) a brief identification of the general thrust of the principal arguments presented to the examiner,
   (The identification of arguments need not be lengthy or elaborate. A verbatim or highly detailed description of the arguments is not required. The identification of the arguments is sufficient if the general nature or thrust of the principal arguments made to the examiner can be understood in the context of the application file. Of course, the applicant may desire to emphasize and fully describe those arguments which he or she feels were or might be persuasive to the examiner.)
6) a general indication of any other pertinent matters discussed, and
7) if appropriate, the general results or outcome of the interview unless already described in the Interview Summary Form completed by the examiner.

Examiners are expected to carefully review the applicant's record of the substance of an interview. If the record is not complete and accurate, the examiner will give the applicant an extendable one month time period to correct the record.

Examiner to Check for Accuracy

If the claims are allowable for other reasons of record, the examiner should send a letter setting forth the examiner's version of the statement attributed to him or her. If the record is complete and accurate, the examiner should place the indication, "Interview Record OK" on the paper recording the substance of the interview along with the date and the examiner's initials.

This document assumes that the project will be split up into different sections that can be implemented and tested separately for the most part. It can also be done all at once, but this means that the entire thing has to be finished before any testing can be done. While this wasn't too much of an issue in Project 3, it's a BAD idea for Project 4. And yes, the caps are warranted there.

The TLB

Up until now, Nachos has used a page table to translate from a virtual address to a physical address. This project switches from the page table to a TLB. The TLB is (by default) a 4-entry TranslationEntry array; in effect, it's a miniature page table. Nachos uses the TLB to convert from the virtual address that it receives into a physical address in order to actually access Nachos main memory.

The problem with the TLB, just like the page table, is updating it. With a page table, it was simple; whenever there was a context switch, the machine->pageTable pointer was updated to point to the page table of the thread taking over the CPU. A TLB isn't quite as simple. There's only the one single TLB for the entire instance of Nachos, and Nachos itself has no clue what to do when it can't find something in the TLB. When this happens, it throws a PageFaultException.

PageFaultExceptions, just like all of the other varieties, wind up in ExceptionHandler, at the bottom of exception.cc. To begin with, it is only set up to handle SyscallExceptions. It must be modified to handle PageFaultExceptions. Creating a new function keeps things a bit cleaner. Name it something bold and imaginative, like HandlePageFault.

Since a PageFaultException is the result of Nachos being unable to find a particular virtual address in the TLB, handling the page fault must include putting that particular virtual address-physical address translation into the TLB before returning. At the lowest level, this consists of finding that entry in the current thread's address space and placing it in the TLB. Since the page table is indexed by virtual address, it is very simple to find the entry in the address space.

Once found, the translation must be placed into the TLB. This is to be done sequentially, replacing TLB entry 0, 1, 2, 3, 0, and so on. This can be accomplished with a simple counter.

NOTE: A TLB replacement can actually replace the translation needed to get the instruction that is being executed. Don't be surprised if you get back-to-back PageFaultExceptions. So long as no single instruction requires more than four different pages in memory, it will eventually be able to execute.

NOTE: A PageFaultException, unlike a SyscallException, should *not* increment the PC, as the instruction at that location in memory has not been completed.

NOTE: Nachos looks for a particular virtual address in the TLB. It does not have any concept of multiprogramming, so it can't tell the virtual address of one process from that of any other process. Be sure to invalidate the contents of the TLB on a context switch. If you want to get fancy, you can do a check and only invalidate the TLB if the incoming thread is from a different process than the outgoing thread. If they are from the same process, they have the same address space.

NOTE: Any time that the TLB is invalidated, the dirty bits must be checked. If something has been changed, this change must be propagated back to the page table/IPT. Basically, this means that the dirty bit must be set. The data has already been changed in memory, but if the page table/ITP don't know about this, that page in memory could potentially be overwritten.

NOTE: There are several functions (copyin, copyinz, copyout) that use the machine->pageTable pointer by default. When using the TLB, this will always be null. These functions must be modified to work without the machine->pageTable pointer.

Simple Algorithm

A. Get the virtual page required by dividing the virtual address by the page size.
B. Find that entry in the current thread's page table.
C. Copy the virtual and physical page numbers from the page table to the slot in the TLB.
   1. At this point, there is no need to propagate back to the page table, as everything is pre-loaded.

The IPT
===

The IPT is an inverted page table; rather than being indexed by virtual page, it is indexed by physical page. It contains the information about what is located in different pages of Nachos main memory. Nachos knows nothing about an IPT. You must create it from the ground up.

The TranslationEntry class is insufficient for the IPT (and the page table, for that matter). Looking ahead to demand-paged virtual memory, you might consider keeping track of some or all of the following in a new class:

1. Physical Page
2. Virtual Page
3. Page Type (code, data, mixed, etc.)
4. Page Location (main memory, swapfile, executable, etc.)
5. Dirty Bit (whether a page has been modified -- VERY important)
6. Use Bit (Never did find anywhere that this is used...)
7. Read-Only Bit (Never found this being used either)
7. Process ID (Important because there are multiple virtual page Xs)
8. Timestamp (For Nachos LRU page replacement later)

Please note that the above is not a definitive, complete list. You may want to add more to it.

The IPT is supposed to keep track of what is in memory. In order to do this, it needs to be updated any time something is moved into/out of memory. With pre-loading, this occurs only when a new address space is created. When demand-paged virtual memory is implemented, this will occur much more frequently during page faults.

Another Simple Algorithm

A. When allocating memory in the AddrSpace constructor, also set the information in the IPT.
   1. At this point, the timestamp is irrelevant.

Demand-Paged Virtual Memory

==========

Prior to this, everything was pre-loaded into Nachos main memory in the AddrSpace constructor. This is no longer the case. A page in main memory is only to be used when it is actually needed during program execution. Hence a page will only be loaded into main memory on a page fault. There is no more pre-loading of memory. This means that you cannot close the executable when creating a process. Check progtest for this. Since the executable remains open, you must keep a handle on it for later use.

The basic idea of virtual memory is to be able to hold more pages in "memory" without actually adding more RAM. The pages that get pushed out of main memory are copied into a swap file on a hard drive. If they are needed again, they get copied back into main memory.

As referenced in the previous section, the TranslationEntry class is no longer sufficient. The page table for each address space needs to be modified to keep track of where a particular page is; if a page is swapped to disk, the only place that knows about it is the page table for that process. Modify the page table to keep track of the additional information that is needed.

Page replacement also requires additional information in the form of a timestamp (at least for the Nachos LRU method of replacement). Nachos knows nothing about the IPT, or any enhanced TranslationEntry class; you'll need to keep track of the timestamp in the IPT. Since Nachos doesn't know about the IPT, it won't update the timestamp for you. The only real place where you can do so is while handling a page fault. At that point, update the timestamp for each page in the TLB (provided, of course, that said page is valid). Note that this is only an approximation, as four pages could potentially have the same timestamp, even if they were actually used at different times by Nachos. Nachos LRU selects the page with the lowest timestamp (lowest tick count) to replace.

All of this references a swap file; all this is in the case of Nachos is a plain old file that you copy stuff into and out of. See the AddrSpace constructor for an example of reading from a file. You'll need to create and open the swap file when Nachos starts, and close it when Nachos finishes. It must be accessible by all processes, as there is a single swap file per instance of Nachos.

On a page fault, if there is no more space in memory, something will need to be removed to make space. If the page is dirty, it must be copied into the swap file and the page table updated. The easiest way to do this is to use the OpenFile WriteAt function. WriteSegment also works, but it's harder to use. Just copy a page sized chunk from Nachos main memory into the swap file. Note that you will need some way to keep track of where in the swap file a particular page has been placed.

Integration

==========

All three of these sections can actually be done independently; the moment of truth arrives when the time comes to put everything together. In theory, it should be a matter of changing what gets updated, and where information is pulled from.

Integration Steps

---

1. Remove any code that pre-loads memory (generally only in the AddrSpace constructor). You will only be loading into/out of memory on a page fault from here on out.

2. Modify the TLB to update from the IPT on a page fault.
3. Modify the IPT to load from the executable/swap file if something is not in memory.
4. Hope that everything works.

The following is *one* possible algorithm for handling a page fault.

Algorithm of Gigantitude

A. Get the virtual page required.
  1. Ensure that the page is within a legal range.
B. Update the timestamp in the IPT for each valid entry of the TLB.
C. Check to see whether the needed page is in the IPT.
  1. If so, go on to step D.
  2. If not, then the page is either in the executable or the swap file and needs to be loaded into memory.
    a. This in turn means that there must be an available slot in main memory.
  3. Select a page in main memory in which to place the new page.
      a. If there is an unused page, use it.
      b. If there are no unused pages and random page replacement is being used, select a random page.
    c. If there are no unused pages and Nachos LRU page replacement is being used, select the page in the IPT with the lowest timestamp.
  4. Check to see whether the selected page is in the TLB.
    a. If so, set it invalid and propagate the dirty bit back to the IPT.
  5. Swap the page from main memory to the swap file.
    a. Depending on which implementation of the swap file you use, the page may or may not already have a swap file page associated with it.
      b. Write the contents of the page into the swap file from main memory.
      c. Update the page table to reflect the location of the evicted page.
  6. Now that there is an available slot in main memory, copy the needed page into it.
    a. Depending on where the page is stored (executable or swap file), read it from there into main memory.
  7. Update the IPT entry to correspond to the new page (PID, virtual page number, physical page number).
D. Update the TLB with the needed page.
  1. If the entry in the TLB that is to be replaced is valid and dirty, propagate the dirty bit back to the IPT.
  2. Copy the information about the needed page into the TLB (virtual page, physical page).
  3. Set the TLB entry's valid bit to true.
E. Increment the TLB counter used to select an entry to replace.

Remote Procedure Calls
==========

Get used to 'em. They'll be back with friends in Project 4. Armed with this knowledge, make sure to set them up well now; it will save you a great many headaches and sleepless nights later. One *IMPORTANT* thing to note is that while the Nachos packet is a character array, the data you send need not be a string. All of Nachos memory is a character array, so it's fairly simple to copy from one place to the other. How you structure your messages is up to you; there are quite a few ways to do it.

General Hints and Tricks
==========

1. Be careful with your macro guards (#ifdef VARIABLE, etc.). They can make life easier, or make a mess of your implementation.

2. Note that this project spans two directories: vm and network. When compiled from the vm directory, it *should not* use RPCs. Make sure that you don't delete the code you put together in the previous projects. Instead, use a macro (#ifdef NETWORK, or somesuch) to selectively compile one version or the other, depending on where it's being compiled from.

3. Whenever you find yourself doing the same thing many times (especially if you wind up copy/pasting code), put it in a function and make sure it works. Put together a little unit test. You really, *really* don't want to find out that your parsing code doesn't work after copying it to 47 different places in the code.

4. Use DDD. It makes life much easier. Run Nachos from within DDD if you hit segmentation faults/bus errors. When that happens, DDD will halt Nachos, but keep the stack. You can use that to find exactly where the segmentation fault is occurring. This is not the be-all and end-all of debugging, though. It really doesn't help when you get a segmentation fault in realfree under malloc.

5. Nachos packets give you a character array of something like 36 bytes by default. Note that you DO NOT have to send messages as strings.

6. When compiling in the network directory, Nachos gives each instance a PostOffice. Said PostOffice comes with its own thread, which keeps checking for messages. This thread never goes away, so Nachos will never halt on its own unless you explicitly halt it as part of Exit.

7. Archive your work. It can be handy to look at an earlier project. A good way to do this is to stick the code directory into a .tar.gz file once you're done with a project (and after you run gmake clean). Something like this:

```
gtar -cf project1.tar code/
gzip project1.tar
```

This creates project1.tar.gz, which is the code directory in a compacted format. The code directory is still there as well. To untar, run the following command:

```
gtar zxf project1.tar.gz
```

This will re-inflate it into the code directory. It is strongly recommended that you do this somewhere other than in nachos-csci402; if you untar it here, it may overwrite your up-to-date files. Just put it in a temporary directory somewhere.

Nachos Assignment #3: Caching: TLB's and Virtual Memory

Tim Brecht
Computer Science 4321/5421
Due date: Tuesday – November 11, 1997, 2:30 p.m.

The third phase of Nachos is to investigate the use of caching. In this assignment we use caching for two purposes. First, we use a software-managed translation lookaside buffer (TLB) as a cache for page tables to provide the illusion of fast access to virtual page translation over a large address space. Second, we use memory as a cache for disk, to provide the abstraction of an (almost) unlimited virtual memory size, with performance close to that provided by physical memory. We provide no new code for this assignment (the only change is that you need to compile with the "-DVM -DUSE_TLB" flags); your job is to write the code to manage the TLB and to implement virtual memory.

Page tables were used in assignment 2 to simplify memory allocation and to isolate failures from one address space from affecting other programs. For this assignment, the hardware knows nothing about page tables. Instead it only deals with a software-loaded cache of page table entries, called the TLB. On almost all modern processor architectures, a TLB is used to speed address translation. Given a memory address (an instruction to fetch, or data to load or store), the processor first looks in the TLB to determine if the mapping of virtual page to physical page is already known. If so (a TLB "hit"), the translation can be done quickly. But if the mapping is not in the TLB (a TLB "miss"), page tables and/or segment tables are used to determine the correct translation. On several architectures, including Nachos, the DEC MIPS and the HP Snakes, a "TLB miss" simply causes a trap to the OS kernel, which does the translation, loads the mapping into the TLB and re-starts the program. This allows the OS kernel to choose whatever combination of page table, segment table, inverted page table, etc., it needs to do the translation. On systems without software-managed TLB's, the hardware does the same thing as the software, but in this case, the hardware must specify the exact format for page and segment tables. Thus, software managed TLB's are more flexible, at a cost of being somewhat slower for handling TLB misses. If TLB misses are very infrequent, the performance impact of software managed TLB's can be minimal.

The illusion of unlimited memory is provided by the operating system by using main memory as a cache for the disk. For this assignment, page translation allows us the flexibility to get pages from disk as they are needed. Each entry in the TLB has a valid bit: if the valid bit is set, the virtual page is in memory. If the valid bit is clear or if the virtual page is not found in the TLB, a software page table is needed to tell whether the the page is in memory (with the TLB to be loaded with the translation), or the page must be brought in from disk. In addition, the hardware sets the use bit in the TLB entry whenever a page is referenced and the dirty bit whenever the page is modified.

When a program references a page that is not in the TLB, the hardware generates a *TLB exception* (in Nachos a PageFaultException), trapping to the kernel. The operating system kernel then checks its own page table. If the page is not in memory, it reads the page in from disk, sets the page table entry to point to the new page, and then resumes the execution of the user program. Of course, the kernel must first find space in memory for the incoming page, potentially writing some other page back to disk, if it has been modified.

As with any caching system, performance depends on the policy used to decide which things are kept in memory and which are only stored on disk. On a page fault, the kernel must decide which page to replace; ideally, it will throw out a page that will not be referenced for a long time, keeping pages in memory those that are soon to be referenced. Another consideration is that if the replaced page has been modified, the page must be first saved to disk before the needed page can be brought in; many virtual memory systems (such as UNIX) avoid this extra overhead by writing modified pages to disk in advance, so that any subsequent page faults can be completed more quickly.

1. Implement software-management of the TLB. For this, you will need to implement some kind of software page translation, for handling TLB misses. Note that with the compile time flag -DUSE_TLB, the hardware no longer deals with page tables; thus, you need to do something about making sure the

TLB state is set up properly on a context switch. Most systems simply invalidate all the TLB entries on a context switch; the entries get re-loaded as the pages are referenced. For item 2, your page translation scheme should keep track of the dirty and use flags for each page set by hardware in the TLB entry (or you could implement a VAX-VMS like scheme).

2. Implement virtual memory. For this, you will need routines to move a page from disk to memory and from memory to disk. We recommend that you use the Nachos file system as backing store – this way, when we implement the file system in assignment 4, we'll be able to use the virtual memory system as a test case. In order to find unreferenced pages to throw out on page faults, you will need to keep track of all of the pages in the system which are currently in use. A simple way to do this is to keep a "core map", which is basically a reverse page table – instead of translating virtual page numbers to physical pages, a core map translates physical page numbers to the virtual pages that are stored there.

   Note: You will probably find it useful to reduce the size of main memory (in machine.h), to more quickly incur paging behavior and to exercise your code.

3. Think about and evaluate the performance of your system. Cache misses (in this case, TLB misses and page faults) can be divided into three categories.

   1. Compulsory misses are those due to the first reference to a cached item; no matter what, you have to pull each referenced page off disk and put it into memory and into the TLB.
   2. Capacity misses are those due to the size of the cache; if the "working set" of the program is larger than main memory or the number of TLB entries, the program will incur misses. Capacity misses are those that would not occur in an infinite sized cache.
   3. Conflict misses are those due to the replacement policy of the cache. These would not occur if the cache used an "optimal" replacement policy, for the same program running on the same size cache.

   Explain how you would write a set of "useful" user programs that demonstrate both a small and large number of each kind of miss, for both the TLB and paging from disk. In other words, explain how to write one test program that demonstrates a small number of capacity TLB misses, then one that demonstrates a small number of capacity page faults, then one that demonstrates a large number of capacity TLB misses, etc. As an example, both sort.c and matmult.c in the "test" directory demonstrate a large number of conflict misses for most standard paging policies.

   For each test case, explain how you would expect your system to perform, and say how you might improve the performance of your system.

   You will probably find it useful to think about reduced main memory sizes.

RECEIVED
CENTRAL FAX CENTER
DEC 19 2006

REMARKS

The Examiner is thanked for his/her careful and very thorough Office Action. No claims are amended at this time. All rejections are hereby respectfully traversed. Favorable reconsideration of the claims is respectfully requested.

Art Rejections

It is respectfully submitted that Kaiser's teaching does not teach the limitations in the claims of the present application. Specifically, Applicant respectfully submits that what Examiner refers to as a "less extreme page fault" (for example, at page 2 of the present Office action) is not a page fault, but is instead a situation where the address translation of command data is not found in the TLB. This only means that the address translation is missing. It does not mean that the data is not present, and it is not a page fault. Kaiser does not describe this situation as a page fault.

Kaiser does describe a page fault in other circumstances--"When a page is accessed by a processor and it is not in real memory, a page fault interrupt occurs and software brings in the page from disk and maps it to a real page in memory." It is specifically noted that the address translation Examiner refers to as a "less extreme page fault" is not, in fact, a page fault. Address translation is performed before the system page table is checked for the needed data. Without performing address translation first, the memory controller (or graphics processor or host processor) would not know what physical address to look in to find the data.

Kaiser does describe a way to alleviate the need to have the host processor perform these address translations. However, Kaiser explicitly uses the host processor when a page fault occurs. It is respectfully submitted that Examiner's characterization of the address translation as a page fault is incorrect. Because of this incorrect characterization, it is respectfully submitted that Kaiser does not teach or suggest, for example, the limitations of claim 1. This is discussed more fully below, with reference to FIGs 1 and 2 of Kaiser.

Kaiser executes graphics processing functions in a graphics processor (206) located in a memory controller 204 attached to a host processor's 12 bus 14, 16. Kaiser says it works by translating virtual (or effective) addresses contained in command blocks (sent to the graphics processor) into real addresses, so that access is made to real addresses in memory (i.e., system memory 24 of FIG. 2). (See generally the summary, col. 3, lines 50-55.)

To do this, the memory controller of Kaiser uses a Translation Lookaside Buffer (TLB 220) to store recently used address translations. The TLB entries are compared to addresses in the commands sent to the graphics processor. If there is no match, this means the translation from the virtual address to a real address is not in the TLB. In short, it is unknown yet where the data resides, and a correct address translation must be found.

The memory controller must take another action to find the proper real address of the data it needs. It performs a page walk (as described at col. 5, lines 50-55): "Memory controller 204 fetches cache lines from the system page table on an as-needed basis to find an effective real address translation..."

At this point, the memory controller now has the address translation it needs to translate the virtual address sent to the graphics processor into a real address for data stored in system memory 24. Because this was a virtual address, some of the virtual memory will not actually be in system memory, but may instead be on the disk.

If the address is not in system memory, and is instead on the disk, then a page fault occurs.

Hence, what Examiner refers to as a "less extreme page fault" is not a page fault at all. It appears that Examiner is referring to the fact that there is no TLB entry that gives a translation for the virtual address sent to the graphics processor. When no such translation exists, a translation is needed. Hence, the memory controller performs a page walk and "fetches cache lines form the system page table on an as-needed basis to find an effective real address translation." This activity is, respectfully, not a page fault.

Only once that address translation is known can the memory controller actually attempt to access the data. Because it is translating a virtual memory address, some virtual addresses will map to system memory (which Kaiser

does not describe as a page fault) and some will map to the disk (which Kaiser does describe as a page fault). If the data sought, after translation, is found to exist in system memory, then Kaiser does not call this a page fault--it simply fetches the data. If, however, the data is not in system memory and is on the disk, then Kaiser describes handling the page fault. It is noted that Kaiser explicitly requires host processor assistance to handle the page fault.

The advantage of Kaiser is that it can translate these addresses without invoking the host processor. See, for example, col. 4, lines 15-19: "It is an advantage of the present invention that 3D graphics processing may be efficiently accomplished by an auxiliary function processor in a controller connected to the processor bus, without the overhead of the processor translating the addresses in the command blocks to real addresses...."

Kaiser does not, however, remove the host processor from the equation when an actual page fault occurs. Kaiser states the required involvement of the host processor at col. 6, lines 3-29.

Hence, Kaiser does not remove the host processor from page faults. It instead removes the host processor from translating the virtual addresses to real addresses. But after such a translation occurs, the system page table must be checked to find out if the data is actually stored in the system memory (which does not require a page fault) or if the data is on the disc (which does cause a page fault). When a page fault occurs in Kaiser, Kaiser must involve the host processor.

Hence, Applicant respectfully submits that Kaiser does not teach or suggest the claimed limitations of,

1. (original): A computer system, comprising:

a graphics accelerator unit which manages page faulting of texture data invisibly to the host processor.

Since there appear to be no circumstances under which Kaiser manages page faulting of texture data invisibly to the host processor, Applicant respectfully submits that all independent claims 1, 2, 3, 4, 7, 14, 15, 17, 20,

and 22 are distinguished from the cited references. Favorable reconsideration of the claims is respectfully requested.

Because of their dependence on allowable claims, all dependent claims are therefore believed allowable.

Favorable reconsideration is respectfully requested.

Conclusion

Thus, all grounds of rejection and/or objection are traversed or accommodated, and favorable reconsideration and allowance are respectfully requested. The Examiner is requested to telephone the undersigned attorney or Robert Groover for an interview to resolve any remaining issues.

DECEMBER 19, 2006

Respectfully submitted,

Patrick C. R. Holmes, Reg. #46,380
Attorney for Applicant

Customer Number 29106
Groover & Holmes
PO Box 802889
Dallas, TX 75380
Tel: 972-980-5840
fax: 972-980-5841